Hudak defines a *domain-specific language* (DSL) as “a programming language tailored to a particular application domain” [10]. DSLs are *little languages* [1] that “trade generality for expressiveness” [11]. A DSL should enable experts in an application area to program without programming—that is, to express the problems they want the computer to solve using familiar concepts and notations, without having to master the intricacies of programming in a general-purpose language [10,16]. For example, the DSL *pic* (long available on Unix-based computers) enables writers to produce line drawings in documents; they can focus on the layout of the drawings without being required to develop programs in C (the primary general-purpose language used on Unix) [1].

The designers of a DSL must select relevant concepts, notations, and processes from the application domain and incorporate them into the DSL design [10]. Often they approach this task in an ad hoc manner. A goal of our research is to identify ways to approach DSL design systematically. In this paper, we leverage commonality-variability analysis [4] of a domain to help identify the needed language constructs and their semantics; we exploit software design patterns [9] and DSL patterns [6] to introduce the desired variability into the DSL’s implementation.

Fowler classifies DSLs into two styles—external and internal [6]. Although the terminology is relatively new, the ideas are not. An *external DSL* is a language that is different from the main programming language for an application, but that is interpreted by or translated into a program in the main language. The language *pic* exhibits this style. An *internal DSL* transforms the main programming language itself into the DSL—the DSL is *embedded* in the main language [10]. The language Lisp (which was defined in the 1960s) supports syntactic macros, a convenient mechanism for extending the language by adding application-specific features.

The rise in popularity of the Ruby programming language [14] and the associated Ruby on Rails web framework [15] has simulated new interest in DSLs among practitioners [2,8]. Ruby—with its flexible syntax and extensive reflexive metaprogramming facilities [3]—provides a convenient platform for developing internal DSLs.

This paper takes a problem motivated by Bentley’s classic column on “Little Languages” [1], constructing a little language for surveys, explores the DSL capabilities of the Ruby language, and designs an internal DSL for specifying and executing surveys. Section 2 describes the Ruby facilities for constructing internal DSLs. Section 3 analyzes the survey problem domain and designs a simple DSL based on the analysis. Section 4 sketches the design and implementation of the survey DSL processor. Sections 5 and 6 examine this work from a broader perspective and conclude the paper.
For this submission I rewrote the Introduction to a paper I published in 2008. My rewrite uses the same citations with the same numbering I used in the original. I replaced item 6 (a web link to work in progress) with the book the author later published from the material I cited.

9. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.